SOFTWARE DEFINED NETWORKING (SDN) CONTROL PLANE IN EVALUATION FOR RELIABILITY AND FAULT TOLERANCE

Wondatir Teka Tefera[1], Nedumaran Arappali[2]
[1,2]Department of Electrical & Computer Engineering, Kombolcha Institute of Technology, Wollo University, Ethiopia.
Email: [1]wonde155@gmail.com, [2]maran.van@gmail.com

Abstract: Software-Defined Networking (SDN) created an opportunity for solving these long standing problems. Some of the key ideas of SDN are the introduction of dynamic programmability in forwarding devices through open southbound interfaces, the decoupling of the control and data plane, and the global view of the network by logical centralization of the "network brain". While data plane elements became dumb, but highly efficient and programmable packet forwarding devices, the control plane elements are now represented by a single entity, the controller or network operating system. Applications implementing the network logic run on top of the ONOS controller and are much easier to develop and deploy when compared to traditional network. The traditional networking architecture can't accommodate the advance user requirements efficiently. Increase of mobile devices, virtualization, high automation & security, efficient Big-Data management and high quality with variety of services, SDN is a promising architecture. For sure current network is not dynamic if compared to SDN. Though yet SDN, needs some time to mature and industry also need time to synchronize the devices according to it. SDN can also manage optical and wireless networks fruitfully. To understand why SDN will play a critical role in future in shaping various technologies, we have to think what actually SDN providing us apart from all technical advances? SDN gives user or operator a feel of nearness to the network. He won't feel that he is a distinct entity when operating on SDN based networks. Hence it is true to say that future networks will revolve around SDN based networking.

Keywords: SDN Network; ONOS Controller; Protocols; Big-Data; Fault tolerance.

1. Introduction

The simplicity in the Internet's design has led to a tremendous innovation in the Internet, but the network itself remains quite hard to change and surprisingly difficult to manage. The root cause of this problem in a traditional network lies primarily in the complicated control plane running on top of all switches and routers throughout the network. These networking devices are manufactured by different network vendors and used trademarked protocols to control the data plane. In these devices, proprietary firmware on the control plane of the switch determines where packets of data are forwarded by the data plane. Distributed optimization of network control was inherently difficult since control plane was a part of individual network devices. Software Defined Networking (SDN) is a relatively new approach to computer networking which evolved from some preliminary research and work done at UC Berkeley and Stanford University in 2005. SDN introduces a layer of software between bare metal network components and the network administrators who configure and set them. This software layer gives network administrators an opportunity to make their network device adjustments through a software interface instead of having to manually configure hardware and actually physically access network devices giving them a very good control over their networks show in Figure 1

.

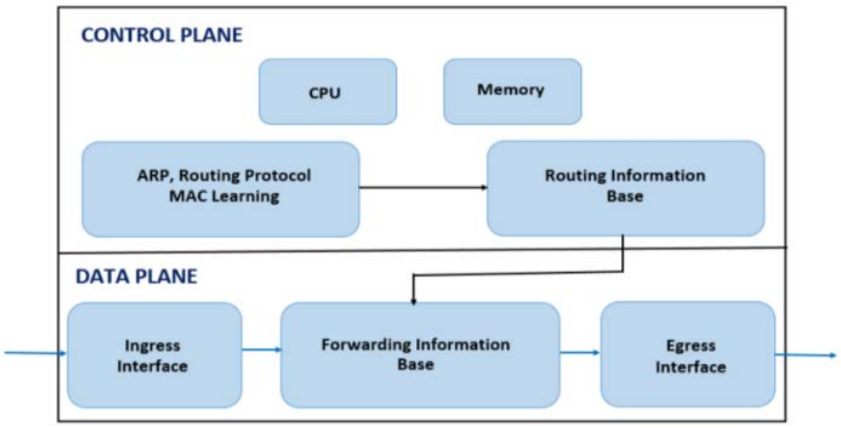

Figure 1 Data and Control plane in traditional networking hardware

This is achieved by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forwardstraffic to the selected destination (the data plane). SDN adheres to open standards and is vendor-neutral, i.e. it can theoretically operate with any vendor's network hardware. This gives organizations the ability to avoid vendor lock-in for a host of network products.

Most current SDN deployments currently rely on a single SDN controller. However, as the number and size of production networks deploying OpenFlow increases, relying on a single controller for the entire network might not be feasible for several reasons. First, the amount of control traffic destined towards the centralized controller grows with the number of switches. Second, since the system is bound by the processing power of that single controller, low setup times can grow significantly as demand grows with the size of the network. This clearly introduces a serious limitation on the scalability and fault tolerance of the controller. The paper aspire to design and build an open source, database backed scalable and fault tolerant OpenFlow controller. The controller is intended to be used for rapid prototyping and research environments.

2. Related work

To evaluate reliability and fault tolerance of control plane architecture we have to review the previous works that have been done related to fault tolerance. We can see these works by dividing in to three sections depending on their control plane scenarios (centralized, hierarchical, and fully distributed) will be discussed as follows.

Logical centralization of the network control plays a central role in SDN architecture. Initial efforts towards implementing such architecture focused on its feasibility in real world scenarios (e.g. campus networks) and on identifying benefits over traditional approaches. Initial designs leveraged the use of single controller deployments, relying on the now-standard OpenFlow protocol. Later, other designs have been developed and released [1, 2, 3]. However, most of them do not provide a very well defined northbound API, nor attempt to shield the application developer from dealing with certain low-level mechanisms of OpenFlow. Recent work provides interesting conclusions interms of centralized controller performance. In [5] show that, with minor code optimizations and a redesign of the controller structure to support multi-threaded parallelism, one can drastically improve throughput and reduce latency on a centralized, single machine controller. Via these optimizations, it has been possible to implement controllers capable to process up to 12 million packet-in messages per second. The research community often argues that, despite the advancements in single controller processing capabilities, it is still not enough to meet the performance, scalability and resiliency requirements of large scale production environments [5]. The intrinsic limitations of this type of design, such as increased latency to forwarding devices in large networks and difficulty in handling large network state, have motivated the development of distributed control plane designs.

Hierarchical control planes are developed around the idea that control instances should have different roles in the control process. Kandoo [6] explicitly separates control among two different layers of controllers. The top layer is composed of a root controller. It is responsible for maintaining global network policies and pushing those down to the bottom layer controllers. The bottom layer is comprised of local controllers. These controllers directly

configure the network forwarding devices, and share their perceived network state with the root controller. This approach has the direct benefit of simplifying the presentation of the Global Network View to control applications at the root controller. However, it requires the development and maintenance of one type of controller for each control plane layer. The root controller introduces concerns as it represents a single point of failure, since it is required to deal with constant network updates. Moreover, it has to operate on an eventually consistent network state, since it can only perceive state that is reported by local controllers. Design is somewhat limited in applicability, being more favorable in low latency environments like data centers, where traffic locality can be well explored in most cases. [7] Presents Difane, a solution for network control in which so-called Authority Switches are assigned the role of forwarding rule caches – much in line with what the local controllers illustrated previously perform in Kandoo.

2.1. General Objectives

The general objective of this thesis is to evaluate reliability and fault tolerance of SDN control plane architecture which is important to customize and use SDN service provider network operating system instead of the current traditional service provider networking systems.

2.2.Specific Objectives

The specific objectives of this paper will be to:

- Select best controller architecture deployment that will be distributed and fault tolerance controller
- Detail analysis on distributed and fault tolerance cluster control plane architecture
- How to coordinate primary and backup controllers
- Simulate and evaluate the fault tolerance of distributed ONOS cluster controller using Mininet emulator.

2.3.Methodology

There are steps to evaluate the reliability and fault tolerance of SDN control plane. These steps are important to make this thesis successfully. The first step will be understood the principles and features of SDN comparing with the traditional network existing on the time being now. The second step will be analysis of Openflow SDN controller platform focusing on their control plane architecture and review different Openflow controller based on their control plane architecture (centralized and distributed). And then select the best controller that has distributed control plane architecture, it will be reliable and fault tolerant. Finally analyze the selected SDN controller with respect to its reliability and fault tolerance by simulating its fault tolerance and reliability using simulation software.

2.4.Architecture of Software Defined Networking (SDN)

An SDN can be logically divided into three different layers. The infrastructure layer refers to the actual forwarding hardware. This layer consists of network devices such as Layer 2 switches in a LAN centric environment. The control layer, also known as the SDN controller is where the real intelligence of a Software Defined Network is situated. This layer implements the basic network services which can be used by various networking applications in the application layer. The switches that are located in the infrastructure layer are not traditional network switches. These switches need to support some mechanism whereby the control layer can talk to and program the switches in the infrastructure layer.

In SDN architecture, southbound application program interfaces (APIs) are used to communicate between the SDN Controller and the switches of the network. They can be open or proprietary. The most popular and well known southbound interface is the OpenFlow protocol.

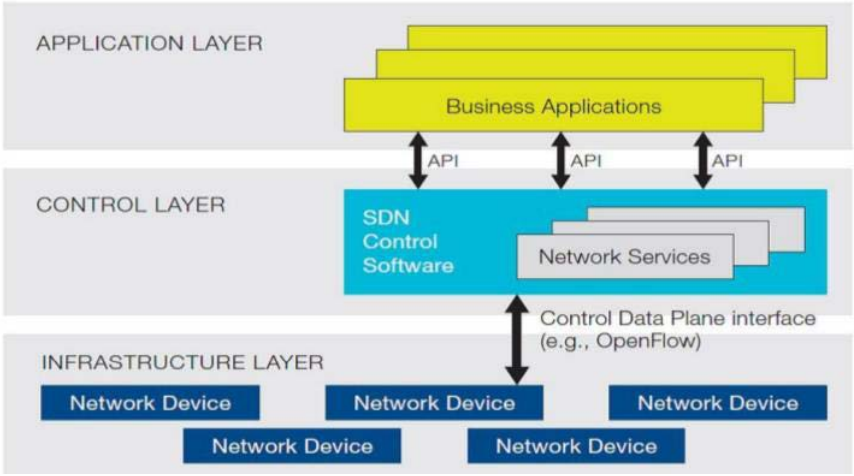

Figure 2. Architecture of a Software Defined Network

The northbound application programming interface (API) on a SDN control layer enables application layer to program the network and request services from it shows in Figure 2. The Northbound API is evolving rapidly but currently there are no standards for it. Each OpenFlow controller provides their own set of interfaces.

How networks are currently structured and operated poses a significant financial issue to Internet service providers and, in fact, has become a handicap for progress in the cloud and service provider space. SDN [13] enables a programmable network control and offers a solution to a variety of use cases. The success stories of these bottom-up SDN solutions have led to a shift in the way operators and vendors perceive the network. In the following, we define four basic principles of SDN. Each of these principles is mandatory for classifying a technology as SDN.

The physical separation of control- and forwarding- or data plane is the best known principle of SDN [14, 15]. It postulates the externalization of the control plane from a network device to an external control plane entity often called the “controller”. In particular, this means that an internal software control plane, while it may still exist, is not enough to brand a device or technology as “Software Defined Networking”. The external controller has to have the ability to change the forwarding behavior of the network element directly. This enables several key benefits of SDN. Control- and data plane can be developed separately from each other, which lowers the entry-to-market barrier, as a company no longer has to have expert knowledge in both areas. Moreover, the externalization of a software-based controller produces pressure on established hardware switch vendors, which are reduced to providing forwarding hardware only. This has already introduced new and disruptive start-ups to the market that have speed up innovation in the network. Even the market leader Cisco has reacted to this trend by introducing its own flavor of SDN with the Application Centric Infrastructure concept developed at the Spin-In company “Insieme” [16]. Customers are also enabled to “mix-and-match” products of different vendors and thus increase competition further. The switch vendors have reacted to that shift by forming the OpenDaylight project for an open SDN software platform. Challenges in this area are to find the appropriate control protocol for the specific scenario out of different protocols and protocol versions, and the appropriate forwarding elements which support this protocol.

The fundamental paradigm shift in networking caused by SDN is represented by the introduction of network programmability. This is enabled by the external software controller and the open interfaces. The programmability principle is not limited to introducing new network features to the control plane but rather represents the ability to treat the network as a single programmable entity instead of an accumulation of devices that have to be configured individually. SDN can thus be regarded as a very suitable complement to network virtualization providing the control plane for an easy operation („programming“) of, e.g., virtual networks in network substrates or to control specific flows within a virtual network as possible applications. Here it is essential to find the appropriate abstraction level, which determines on the one hand the

ease-of-use for network programmers and on the other hand the abstraction overhead and there with possible performance degradation.

2.5. The OpenFlow Architecture

OpenFlow(OF) is considered the pioneer SDN standard. This protocol enabled the controller to interact directly with the underlying devices (both physical and virtual), making the SDN adapt to changing application requirements. The controller could set rules about forwarding behaviors of each device through the OF protocol like modify, drop, en-queue and forward a packet belonging to particular flow. The working of the OF protocol can be explained with the Figure 3.

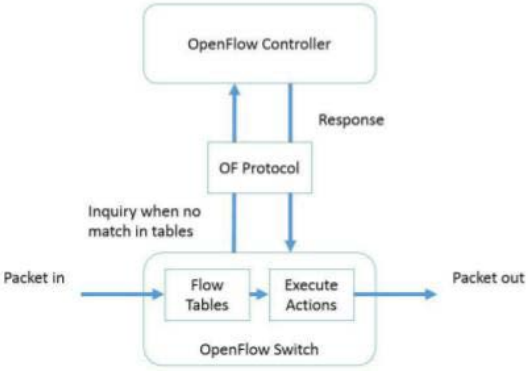

Figure 3 - OpenFlow Architecture

Every switch in the network has its own set of flow tables. Furthermore these tables consist of flow entries. A flow entry can be seen as the forwarding/routing rules. It has three key components: a bit pattern indicating the flow's properties, a list of commands, and a set of counters. Whereas a flow is the set of packets that match a particular flow entry. Figure 2.5(a) displays the main components of an OpenFlow-based network and the decision making process an incoming packet goes through. Thus when a packet arrives at the switch it can undergo one of the following scenarios:

1. The packet is a match to a flow entry in the switch's set of flow table and is then forwarded according to the rules of that particular entry.

2. The packet is a match but there are no actions associated with that flow entry, thus the packet is dropped.

3. The packet is not a match, and is queued with an inquiry being sent to the controller, to which the controller replies with a new OpenFlow entry resulting in future packets to be handled by the switch itself.

As mentioned in the above section OpenFlow protocol is the most popular and widely accepted protocol for the southbound Application Programming Interface. OpenFlow protocol intends to provide access to the data plane of the switches. It does this by specifying a language that a switchcan recognize and use to update its forwarding tables. OpenFlow is a language for generically defining characteristics of a particular flow of traffic and a set of actions to be executed when the switch encounters packets that matches such characteristics. The actual mechanisms used to program flows into switch hardware very greatly depending on the vendor of the particular hardware. Instead, OpenFlow provides a way to describe desired flow state within an agent running locally on the forwarding device. All switches that are OpenFlow enabled will have the OpenFlow agent that will interpret the OpenFlow commands. The OpenFlow specification also includes ways for the OpenFlow controller, which is remote and located in the control plane to make modifications to this information. The OpenFlow agent, armed with the flow information programmed into it by a controller, acts like the control plane on traditional switches. The only difference is that it does not have to run routing protocols, or make decisions locally. All the decisions are made by the remote OpenFlow controller and the OpenFlow agent stores these OpenFlow entries, and pushes them into the flow tables on the hardware device. Figure 4 shows an idealized OpenFlow switch where the flow table is controlled by a remote OpenFlow controller

.

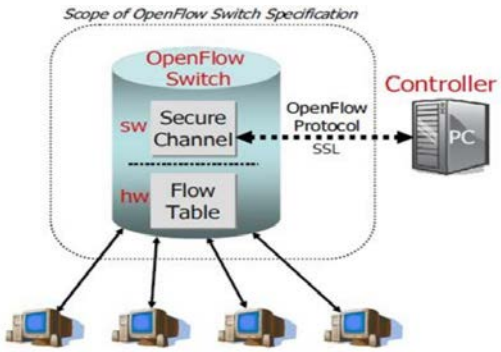

Figure 4 - Idealized OpenFlow Switch

The OpenFlow controller has a unified view of the whole network. It runs the routing or switching protocols to collect the relevant routing or switching information. There are two different ways in which the OpenFlow controller can program the switches in the network.

The first packet of each new flow can trigger the controller to insert flow entries down to the switches and the switch makes efficient use of flow table where every flow needs small additional flow setup time. The other approach is that the flow tables in switch can be pre populated by the OpenFlow controller ahead of time for all traffic matches that could come into the switch. By predefining all the flows and actions ahead of time in the switch flow tables, the packets can beforwarded at line rate as this approach does not require any additional flow setup time per individual flow.

The interface that connects each network device to a controller is named the OpenFlow channel. This is the interface that is used by the controller to manage and configure the underling switches, and vice versa receiving messages from the switches. Similarly the switches use it denote a packet arrival, switch state change or any update e.g., alarms. Thus the OF protocol messages can be categorized into three types [18].

- Controller to Switch Messages

  These types of messages are initiated by the controller and are employed to directly manage, configure or inquire status of the devices underneath.
- Asynchronous Messages

  These are the messages initiated by the switches (without the controller asking) to notify a packet arrival, to give error messages or any changes in their state.
- Symmetric Messages

  These can be initiated both by the switch and the controller and could be used for sending connection establishment messages (hello/echo messages) or for testing latency.

There are two types of approaches used by the switches to deal with a packet. Either switch forwards to controller a message called Packet. In which contains complete packet or sends some information of the header to get routing information. In this case the switch needs to have a buffer to store the packet otherwise it gets discarded. The variable packet sizes can be observed in open flow channel. This packet size can vary from 66 Bytes Hello message to 1500 Bytes data packet. The symmetric messages are sent periodically to check the existence of device. The frequency of routing queries depends on flow arrivals, departures and idle flow removals.

3. Result and Discussion

3.1. OPEN NETWORK OPERATING SYSTEM (ONOS)

As we have seen trade-offs of the control plane architecture of OpenFlow controllers in chapter two, ONOS meets the optimum requirements of trade-offs between correctness, availability, and consistency by having both eventually and strong consistency model algorithms. It is important to select ONOS as service provider network operating systems. ONOS is built to provide high availability (HA), scale-out, and performance for these networks demands. The Southbound modules manage the physical topology, react to network events and program/configure the devices forcing on different protocols. The Distributed Core is responsible to maintain the distributed data stores, to elect the master controller for each network portion and to share information with the adjacent layers. The NorthBound modules offer an abstraction of the network and

the interface for application to interact and program the NOS. Finally, the Application layer offers a container in which third-party applications can be deployed.

ONOS provides basic platform for distributed SDN scenario. Architecture of ONOS has been shown in Figure 5.

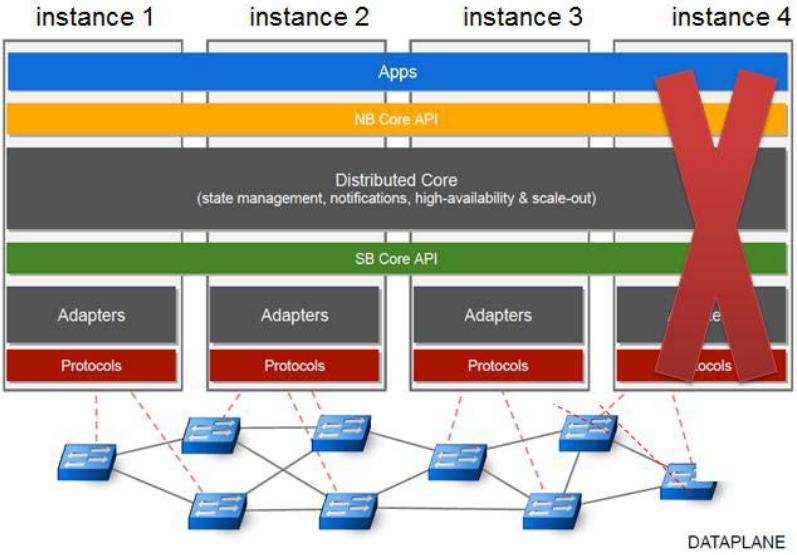

Figure 5- ONOS Distributed Architecture

ONOS platform is a multi-module project based on instance of Apache Karaf in which modules are managed as OSGi bundles. Apache Karaf also provides local and remote access to ONOS control and a CLI interface. Let's discuss the components shown in Figure 5.

Network Applications -These applications are used to facilitate management across all ONOS instances and to enforce policies in the network e.g., QoS, Resource allocation etc. It provides flexibility by allowing developers to make their own custom application suite.

Northbound APIs - These APIs provide abstract view of the network to application layer. They provide information related to network topology, devices and links. They also provide application intent framework to network applications. The application intent framework allows application to specify high level intents without underlying rules i.e they need only directions what to do rather than how to do". The intent framework allows an application to request a service from the network without having to know details of how the service will be performed. This allows network operators as well as application developers to program the network at high level; they can simply specify their intent: a policy statement or connectivity requirement. Some examples of intents:

- Set up a connection between host A and host B
- Set up an optical path from switch X and switch Y with z amount of bandwidth
- Don't allow host A to talk to host B

The Intent framework takes such requests from all applications, figure out which ones can and cannot be accommodated, resolves conflicts between applications, applies policies set by an administrator, programs the network to provide the requested functionality, and delivers the requested services to the application. The Global Network View provides the application with view of the Network – the hosts, switches, links, and any other state associated with the network such as utilization. An application can program this network view through APIs. One API lets an application look at the view as a network graph. Some examples of what can be done with the network graph include:

- Create simple application to calculate shortest paths since the application already has a graphical view of the network.
- Maximize network utilization by monitoring the network view and programming changes to paths to adjust load (traffic engineering).

Distributed Core - This is the core of ONOS architecture. Its functions include management of topology stores, provision of consistency, availability and cluster state management.

3.2.Environment setup

Mininet is a lightweight container orchestration system for network emulation. With Mininet and onos.py, it can easily start up an ONOS cluster, and a modeled data network for any topology you might like, in a single VM or server. This is usually the most convenient way to create an ONOS development environment on memory space constraint laptop and it can be up and running in a matter of minutes (or seconds if have already built ONOS and have already installed Mininet).

Running Mininet on a laptop, use onos.py to start up a complete emulated ONOS network in a single VM - including ONOS cluster, modeled control network, and data network. This simplifies development on a laptop, because it can run a single development VM (or no VM at all since it runs on a Linux machine). Moreover, it is more efficient than a multi-VM setup because the entire emulated network lives in a single VM and shares a single Linux kernel. Additionally, onos.py models the control network as well as the data network; it is easily to change the number of nodes in the ONOS cluster, as well as things like the delay or bandwidth between nodes in the control network. It's even possible to change the control network topology as well as the data network topology. onos.py provides a single, unified console via the mininet-onos> CLI, where it is possible to enter both Mininet and ONOS commands – this can be very convenient. onos.py also automatically handles port forwarding , so it can easily connect to the GUI (or to karaf, or to the controllers' OpenFlow ports) by connecting to ports on the VM. onos.py parametrizes both the control network (ONOS cluster) and the data network; so it's easy to restate over multiple cluster sizes and network topologies.

The basic setup used for the simulation of scalability and fault tolerance is described here. There are three servers (ONOS instances) “onos1”, “onos2” and “onos3” running three ONOS cluster OpenFlow controllers. All of these OpenFlow controllers are backed by the same Titan Graph database running on the same server. The machines onos1, onos2 and onos3 have IP addresses 192.169.123.1, 192.168.123.2 and 192.168.123.3 respectively. As mentioned above mininet is used to simulate the network topology. The mininet runs on the ONOS server running on host computer. The topology used for simulation contains 21 switches and 64 host devices. The switches are numbered sequentially as s1, s2, s3, s4, s5, s6 and so on. The hosts connected to those switches are similarly numbered as h1, h2, h3, h4, h5, h6 and so on. The hosts have IP addresses assigned from 10.0.0.0/24 subnet with the last octet representing their host number. For e.g. h1 will have an IP address of 10.0.0.1, h2 will have an IP address of 10.0.0.2 and so on. The network topology that has 64 host and 21 switches is running by using the following command show in Figure 6.

sudomn --custom onos.py --controller onos,3 --topo tree,depth=3,fanout=4

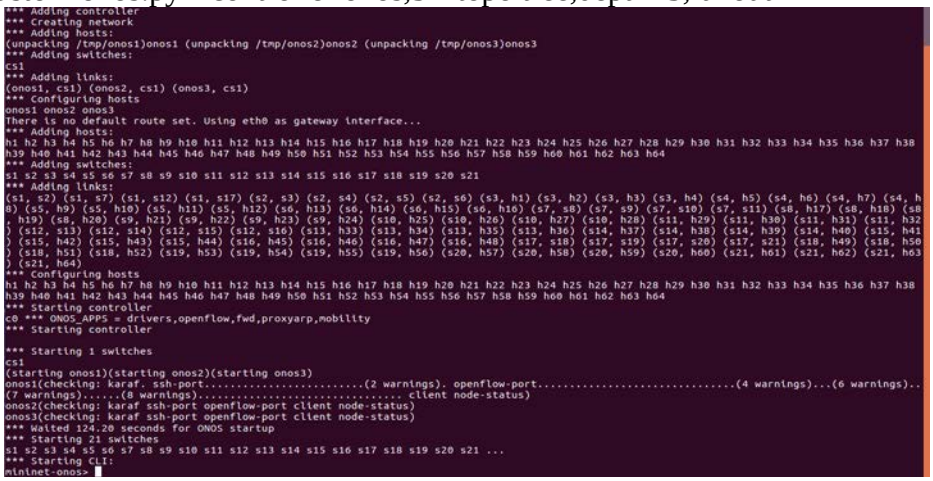

Figure 6 representation of the network topology

On start-up all the switches in the topology connects to all ONOS OpenFlow controller instances. Even though all the switches connect to all the controllers, each switch in the topology has one of the controllers as the master controller. This is done by per switch master election using ovs-vsctl set-controller command. onos:balance-mastersONOS controller has been run by typing onos on the mininet CLI as shown the figure 7 on below

Figure 7.Running ONOS command window

From the onos CLI can check how many onos instances (nodes) running by using the command nodes as shown below.

Figure 8. Running 3 ONOS nodes

The above figure8 shows three nodes of onos instance running on the same ONOS server and their ip addresses are 192.168.123.1, 192.168.123.2 and 192.168.123.3 respectively.

And cluster of onos instances can be displayed by using GUI as shown below. Color key indicates the control mastership mapping of forwarding devices with the corresponding Control instance node.

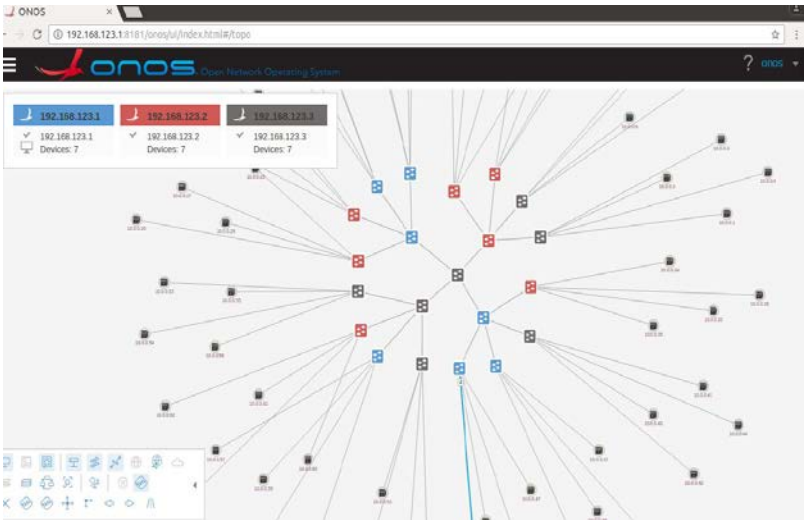

Figure 9. The nodes ONOS GUI representation

The above figure9 shows that all switches are connected to one onos instance as a master instance and it can be balance the load that mean distributed the switch for all onos1, onos2 and onos3 controllers by using onos:balance-masters that both instances connected with the corresponding color switches as shown the figure10.

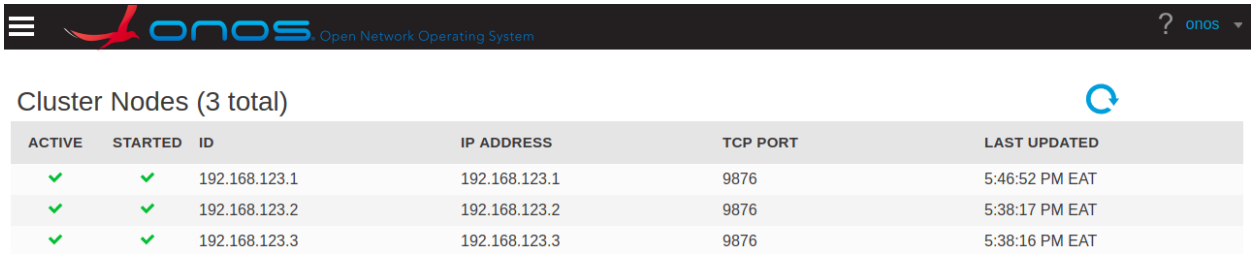

Figure 10. ONOS cluster nodes before the master controller fails

To test fault tolerance by downing onos1 using the following command

onos>onosdown onos1

Bringing onos1 down…

After downing onos1 the entire network is controlled by onos2 and onos3 instances as shown from GUI representation of figure11 below.

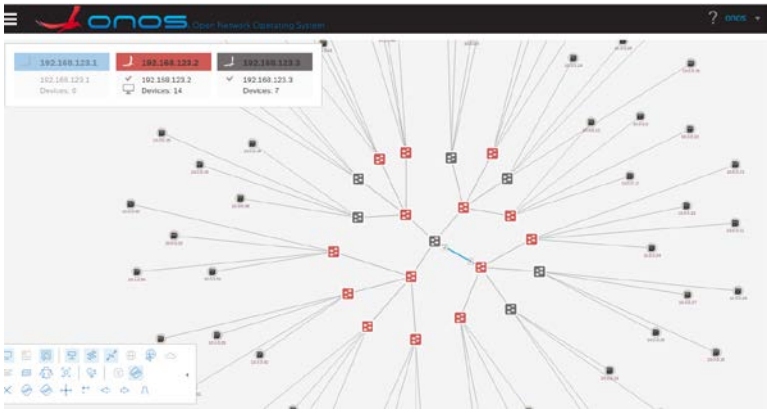

Figure 11. Fault tolerance test displayed on GUI

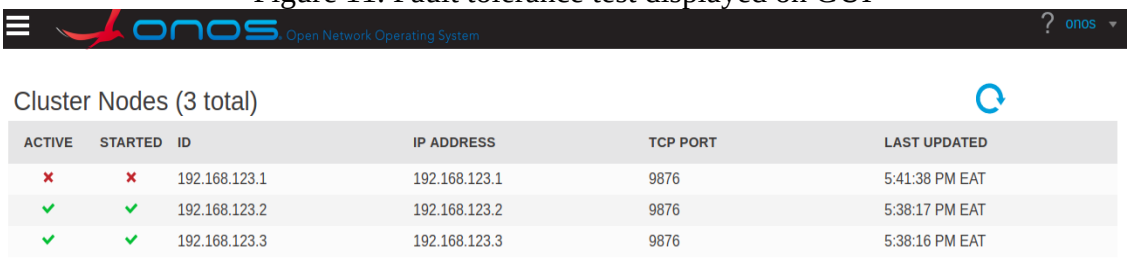

Figure 12. ONOS cluster nodes after the master controller fails

To know the role of the three controllers for each switch in the network on the command window show in Figure12.

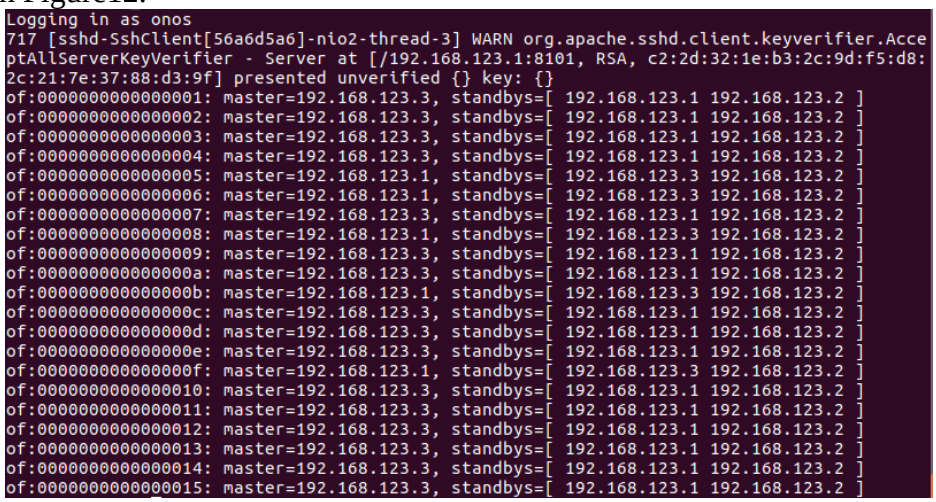

Figure 13. Role of cluster controller in the network

As we can see the above figure13 for each device in the network topology has their own one master and two backup controllers; this indicates that when the master controller fail one of the backup controller will be master and resume the task of the failure controller with the up dated data state.

Leader election for each term of task the leader is elected in the cluster member of onos instances using the raft algorithm leader election procedure that means the most up to-

date data long term is elected as master controller .this can be shown by the following figure.

Topic	Leader	Term	Elected
device:of:0000000000000004	192.168.123.1	2	8s ago
device:of:0000000000000005	192.168.123.1	1	27m ago
device:of:0000000000000006	192.168.123.1	1	27m ago
device:of:0000000000000008	192.168.123.1	1	27m ago
device:of:000000000000000b	192.168.123.1	1	27m ago
work-partition-5	192.168.123.1	2	27m ago
work-partition-6	192.168.123.1	2	27m ago
device:of:000000000000000f	192.168.123.1	1	27m ago
work-partition-4	192.168.123.1	2	27m ago
work-partition-7	192.168.123.1	2	27m ago
work-partition-8	192.168.123.1	2	27m ago
device:of:0000000000000014	192.168.123.1	2	8s ago
device:of:0000000000000010	192.168.123.2	2	8s ago
device:of:0000000000000011	192.168.123.2	2	8s ago
device:of:0000000000000003	192.168.123.2	2	8s ago
device:of:000000000000000a	192.168.123.2	2	8s ago
work-partition-1	192.168.123.2	3	27m ago
work-partition-2	192.168.123.2	3	27m ago
device:of:000000000000000c	192.168.123.2	2	8s ago
device:of:000000000000000d	192.168.123.2	2	8s ago
work-partition-0	192.168.123.2	3	27m ago
device:of:000000000000000e	192.168.123.2	2	8s ago
work-partition-3	192.168.123.2	3	27m ago
work-partition-12	192.168.123.3	1	27m ago
work-partition-13	192.168.123.3	1	27m ago
device:of:0000000000000001	192.168.123.3	1	27m ago
device:of:0000000000000002	192.168.123.3	1	27m ago
device:of:0000000000000007	192.168.123.3	1	27m ago
device:of:0000000000000009	192.168.123.3	1	27m ago
work-partition-9	192.168.123.3	1	27m ago

Figure 14. Snapshot of how leader elects for each term

As we can see from the figure14 above on the Topic column lists are the work partition and devices that controlled by the elected controller, on the term column the listed number indicates the log term sequence that the up-to-date data log entry number and the last column indicates the timestamps that the leader elected.Communication of the cluster node controllers can be checked by using network protocol tools called wireshark and we can show figure15 the packet transfer among the three ONOS nodes as follows

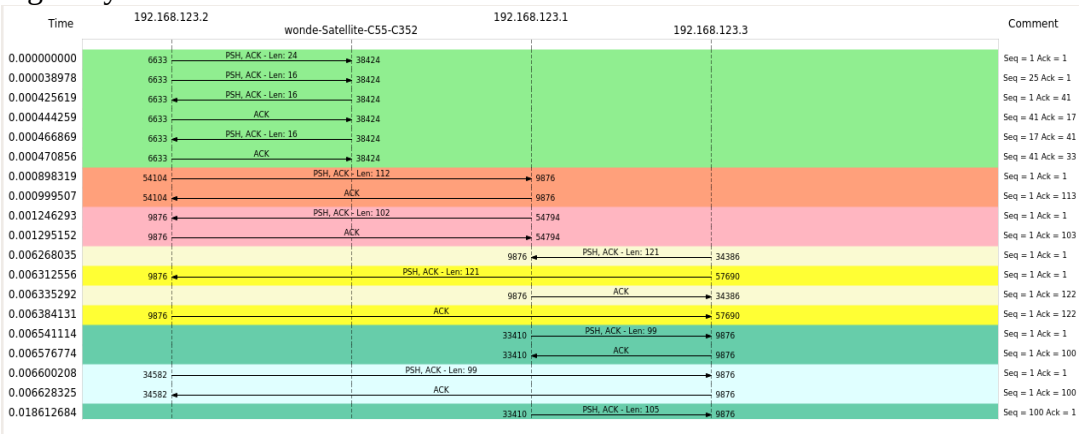

Figure 15. Packet transfer among the three nodes of ONOS cluster using wireshark

As we can see from the above figure the port number between the controllers is 9876 and to connect the data plane the controller uses 6633 port.

4. Conclusion

Traditional networks are complex and hard to manage. One of the reasons is that the control and data planes are vertically integrated and vendor specific. Another, concurring reason, is that typical networking devices are also tightly tied to line products and versions. In other words, each line of product may have its own particular configuration and management interfaces, implying long cycles for producing product updates (e.g., new firmware) or upgrades (e.g., new versions of the devices). All this has given rise to vendor lock-in problems for network infrastructure owners, as well as posing severe restrictions to change and innovation.In this paper was discussed the importance of having a distributed SDN network, in order to avoid the problem of having a single point of failure and also because with a distributed system is possible to achieve high availability, scalability and, persistency of the information. Was also analyzed the behavior of the ONOS controller under the cluster architecture schema, with the aim of having a better understandability of how is acting the controller. One of the biggest parts of the work was theresearch for the protocols that were being used for the network in the communication between controllers, in this part was also explained the different messages

exchanged between controllers. This was performed with help of Wireshark, making traffic captures along the interfaces involved and analyzing the packets with the aim of understand the communication.

References

1) MCCAULEY, J. POX: A Python-based OpenFlow Controller.
2) http://www.noxrepo.org/pox/about-pox/. 2014. Available from Internet:
3) ERICKSON, D. The beacornopenflow controller. In: ACM. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. [S.l.], 2013. p. 13–18.
4) Ryu SDN Framework Community. RYU network operating system. 2015. Available from Internet:
5) SoftwareDefinedNetworking:TheNewNorm forNetworks.Available:https://www.opennetworking.org/images/stories/downloads/sdn-resources/whitepapers/wp-sdn-newnorm.pdf
6) TOOTOONCHIAN, A.; GANJALI, Y. HyperFlow: A Distributed Control Plane for OpenFlow. In: USENIX INM/WREN. [S.l.: s.n.], 2010.
7) YEGANEH, S. H.; GANJALI, Y. Beehive: Towards a Simple Abstraction for Scalable Software-Defined Networking. In: Proceedings of the 13th ACM Workshop on Hot Topics in Networks. New York, NY, USA: ACM, 2014. (HotNets-XIII), p. 13:1–13:7. ISBN 978-1-4503-3256-9. Available from Internet.
8) YU, M. et al. Scalable Flow-based Networking with DIFANE. In: ACM SIGCOMM. [S.l.: s.n.], 2010.
9) A. R. Curtis et al. DevoFlow: scaling flow management for high-performance networks. ACM SIGCOMM CCR, New York, NY, USA, v. 41, n. 4, 2011. ISSN 0146-4833
10) LEVIN, D. et al. Logically centralized?: state distribution trade-offs in software defined networks. In: HotSDN. [S.l.]: ACM, 2012. ISBN 978-1-4503-1477-0.
11) Amin Tootoonchian, YasharGanjaliHyperFlow: A Distributed Control Plane for OpenFlow
12) TeemuKoponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievskiy,MinZhuy, Rajiv Ramanathan, YuichiroIwataz, Hiroaki Inouez, Takayuki Hamaz, Scott Shenker. Onix: A Distributed Control Platform for Large- scale Production Networks.
13) H. Kim et al. CORONET: Fault tolerance for software defined networks. In IEEE ICNP, 2012.
14) T. D. Nadeau and K. Gray, SDN: Software Defined Networks. Media, 2013.
15) M. Jarschel, F. Wamser, T. Höhn, T. Zinner, and P. Tran-Gia, “SDN-based Application-Aware Networking on the Example of YouTube Video Streaming,” in Proceedings of the 2nd European Workshop on Software Defined Networks (EWSDN 2013), Berlin, Germany, October 2013, pp. 87–92.
16) J. Duffy, “Cisco takes fight to SDNs with bold Insieme launch,” http://www.networkworld.com/news/2013/110613-cisco-insieme- 275666.html, November 2013, last accessed on 2014.03.06.
17) G. Hampel, M. Steiner, and T. Bu, “Applying software-defined networking to the telecom domain,” in Proceedings of 16th IEEE International Global Internet Symposium (GI 2013) collocated with IEEE INFOCOM 2013, Turin, Italy, April 2013, pp. 133–138.
18) OpenFlowSwitchSpecificationv1.3.1.Av at:https://www.opennetworking.org/images/stories/downloads/sdnresources/onfspecifications/openflow/openflow-spec-v1.3.1.pdf
19) B. et al. Pfaff. OpenFlow Switch Specification Version 1.3.0 Implemented (Wire Protocol 0x04).J. O. Diego Ongaro, "In Search of an Understandable Consensus Algorithm," Stanford University, 2014.
20) J UNQUEIRA , F. P., R EED , B. C., AND S ERAFINI , M. Zab: high-performance broadcast for primary-backup systems. In Proc. DSN‟11, IEEE/IFIP Conference on Dependable Systems and Networks (2011), IEEE, pp. 245–256. 2, 137, 149.